Zero-Trust Software-Defined Vehicles: a Security Paradigm for the Automotive Cloud-Edge Continuum

Giovanni Lombardo[1,*], Vincenzo Alessio Bucaria[1], Giovanni Merlino[1,2] and Francesco Longo[1,2]

[1] *University of Messina, Department of Engineering, Contrada di Dio, 98166 Sant'Agata, Messina, Italy*
[2] *National Interuniversity Consortium for Informatics (CINI), Via Ariosto 25, 00185 Rome, Italy*

Abstract

The automotive sector is undergoing a fundamental transformation as vehicles evolve from hardware-centric systems into highly connected, software-defined platforms operating within a broader digital computing continuum — spanning fog, edge, and cloud resources. In such a dynamic yet threatening environment, we discuss the necessity of a robust security model, enabling flexible execution, strong application sandboxing, and integrity verification of both the runtime and application modules, and we propose the Zero-Trust Software-Defined Vehicle paradigm. After introducing the core concepts and motivations, we analyze relevant use cases, main requirements, and future research directions. Identifying potential enabling technologies, we also present some preliminary experimental results highlighting the need for challenging yet exciting further investigation.

Keywords

Software-defined, Zero-Trust, Cloud-Edge Continuum, Automotive, WebAssembly, Safety

1. Introduction

The automotive sector is undergoing a fundamental transformation as vehicles transition from hardware-centric machines to highly connected, software-driven platforms. The increasing quantity of on-board available computational resources is transforming modern cars into distributed network of embedded devices that support both core functionalities and customizable updates. This setup enables the possibility of distributed applications deployment that are tightly integrated with the vehicle's sensory and actuation subsystems, delivering advanced functionalities for users (drivers, passengers), optimizing in-vehicle operations, and enabling onboard execution for complex scenarios.

This evolution has led to the Software-Defined Vehicle (SDV) paradigm, in which vehicle capabilities and services are delivered, updated, and orchestrated primarily via software. Much like smartphones, SDVs can receive new applications or firmware updates that enhance performance, add new features, or adapt to user preferences throughout the vehicle's lifecycle. This shift enables unprecedented levels of personalization, maintainability, and extended product utility. However, increased software control and ubiquitous connectivity come with a proportional escalation in cybersecurity challenges. Modern SDVs expose a broad and dynamic attack surface that spans embedded electronic control units (ECUs), cloud services, edge computing nodes, and mobile or third-party applications. Threats can originate from any layer, internal or external, and may target safety-critical systems such as braking, steering, or acceleration.

Traditional perimeter-based security approaches, which rely on a trusted internal network and static firewalls, are no longer sufficient. The assumption that components within the car can be inherently trusted breaks down in a setting where software is modular, updatable, and potentially authored by third parties. As vehicles increasingly operate as part of a broader digital continuum, including fog, edge, and

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT
*Corresponding author.
lombardog@unime.it (G. Lombardo); vbucaria@unime.it (V. A. Bucaria); gmerlino@unime.it (G. Merlino); flongo@unime.it (F. Longo)
0009-0003-0653-9349 (G. Lombardo); 0009-0008-0837-4070 (V. A. Bucaria); 0000-0002-1469-7860 (G. Merlino); 0000-0001-6299-140X (F. Longo)

cloud resources, the need for a more robust and dynamic security model becomes clear. To address these evolving threats, we propose the Zero-Trust Software-Defined Vehicle (ZT-SDV) paradigm. Grounded in the Zero-Trust principle of "never trust, always verify", ZT-SDV treats every component, application, user, or external service as untrusted until authenticated, authorized, and continuously validated. This paradigm ensures that each data exchange and control action is cryptographically verified regardless of its origin, location, or context. Several enabling technologies can support the diverse requirements of our proposal. In this work, WebAssembly (WASM) and Trusted Execution Environments (TEE) are presented as candidate technologies for achieving strong application sandboxing and flexible execution, together with integrity verification of both runtime and application modules. In addition, the end user always retains final control; by leveraging mechanisms such as reproducible builds, he can verify the origin and trustworthiness of the code before executing it.

The remainder of the paper is structured as follows. Section II presents and explains the main background concepts that help the reader understand the foundation of this work. Section III focuses on the definition of Zero-Trust Software-Defined Vehicles, along with possible use cases. Section IV analyzes related works. Section V introduces the requirements necessary for the correct functioning of the system. Section VI analyzes the enabling technologies to meet these requirements. Section VII presents possible research trajectories together with some preliminary performance results of I/O driving using WASM. Finally, Section VIII concludes the paper and outlines more general directions for future research.

2. Context and point of view

This section outlines the background in which the concepts presented in the paper originated and evolved. It offers a perspective that helps the reader better understand the motivations underlying the formulation of the Zero-Trust Software-Defined Vehicle paradigm.

2.1. Cyber-Physical Systems

The increasing integration between hardware and software in modern engineered systems requires a profound alignment between their cyber and physical components. This conceptual convergence has given rise to a systems-oriented approach to Cyber-Physical integration.

A Cyber-Physical System (CPS) merges computational elements with physical processes, where overall behavior arises from the combined influence of cyber and physical subsystems. Embedded computers and networks oversee and regulate these physical processes through feedback mechanisms, enabling physical states to shape computations while computational outputs drive physical changes. Conceptually, CPS emphasize the intersection rather than the simple combination of physical and cyber aspects and their reciprocal interactions define the system's core functionality. In this context, vehicles represent a prime example of CPS because they embody a tight integration of physical processing devices, such as engines, brakes, sensors, and actuators, with cyber elements like electronic control units (ECUs), autonomous-driving algorithms, and safety functions.

This definition is intrinsically embedded within our vision, as it delineates how software and parameter updates manifest as tangible changes in vehicle behavior, accommodating diverse use cases and operational scenarios.

2.2. Computing Continuum

The cloud is commonly seen as a virtually inexhaustible source of computational resources in which applications can be deployed and elastically scaled to accommodate diverse usage scenarios. However, contemporary cloud environments typically encompass both public and private cloud infrastructures, are frequently geo-distributed across multiple regions, and may additionally incorporate components and computing capabilities located at the edge of the network. For this reason, modern cloud infrastructures exhibit a high degree of structural and operational complexity [1]. The concept of cloud continuum

can be viewed as a seamless integration of diverse cloud computing environments that collectively form an interconnected ecosystem, including data centers, private, public, hybrid, and multi-cloud infrastructures. It enables flexible deployment and management of computation and data across centralized clouds, distributed edge, and on-premise resources, optimizing for factors like latency, security, cost, and scalability.

The modern concept of the connected vehicle fits naturally within the cloud continuum [2]. At the lowest level, the vehicle itself represents the edge, composed of heterogeneous sensors, actuators and electronic control units (ECUs) that implement safety-critical functions (e.g., automatic braking) strictly on-board to satisfy tight real-time constraints. Beyond the vehicle, roadside and infrastructure units aggregating data from multiple vehicles and processing it to enable services such as traffic management, inter-vehicle communication and adaptive speed control form the fog layer. Finally, computationally intensive tasks without strict timing constraints, such as large-scale data analytics, AI model training and global fleet monitoring, can be offloaded to the cloud layer.

2.3. Software-Defined X

In contemporary large-scale infrastructures such as communication networks, the rapid growth and increasing rigidity of data centers have highlighted the need for more flexible, programmable, and easily adaptable solutions. This has led to the emergence of the Software-Defined X (SDX) paradigm, which considers software as the primary determinant of system behavior and reduces the reliance on specialized hardware. Through the introduction of suitable abstraction layers, the software-defined approach seeks to clearly separate control intelligence from the execution or data plane, thereby decoupling the "what" (policies and control) from the "how" (low-level execution and data handling). In this perspective, hardware mainly provides generic computation, communication, and actuation capabilities, while the global system behavior is governed by reconfigurable software components that orchestrate these resources.

The SDX concept initially gained traction in the networking domain with Software-Defined Networking (SDN), where traffic control logic is consolidated into a programmable controller and forwarding elements are simplified into devices that expose standardized interfaces toward that controller. The same principle has been extended to Software-Defined Vehicles (SDV), in which traditional, specialized ECUs are replaced by more powerful and centralized computing platforms that interface with sensors and actuators through common middleware and can be updated over the air throughout the vehicle's lifetime. More recently, this paradigm has been generalized to broader smart environments, such as Smart Cities, giving rise to the notion of Software-Defined Cities (SDC) where the different actors (e.g. municipalities, companies, scientists, and citizens) of this typical heterogeneous ecosystem can easily collaborate in developing innovative smart services [3].

2.4. Software trustworthiness

In the context of cyber-physical systems (CPSs) and especially in modern vehicles, software plays a pivotal role in supporting decision-making and controlling actions. In a very concrete sense, software increasingly assumes the role of the "driver" of the vehicle, which can naturally raise concerns and doubts for end users. Given this central role, a key question for human drivers and passengers is whether such software can genuinely be regarded as reliable. Safety becomes the core concept that underpins software trustworthiness, since software governs the operations that ensure driver safety and, ultimately, protect human life in the event of an accident. The fundamental challenge, therefore, is software trustworthiness.

Software trustworthiness denotes a software system's ability to satisfy user expectations through its behavior and outcomes, while maintaining continuous operation even in the presence of disturbances or failures [4] . The trustworthiness of a software system can be examined in terms of its provenance, construction, and the expectations placed upon it. In particular, users should be able to understand

where the software originates, inspect its source code, and, if necessary, rebuild it in order to verify and reproduce its intended behavior. A trustworthy process is one that can be audited at any point during design, development, and operation to assess the degree to which it can be trusted.

This concept includes different aspects and requirements that will be better analyzed in the following sections, but overall it represents a solid foundation for establishing trust in a well-defined software-driven system.

2.5. Zero-Trust Architecture

The classical approach for network defense consists of the so called "perimeter-based security architecture". In this specification, the network is separated into internal and external zones using tools like firewalls, Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS). Devices inside the internal network are automatically considered trusted, while external ones need authentication before gaining trust. However, once authenticated, an entity remains trusted indefinitely, allowing potential malicious actors continued access [5]. For this reason, the National Institute of Standards and Technology (NIST) introduced the new concept of Zero-Trust-Architecture (ZTA) [6]

Differently from the perimeter-based one, in a Zero-Trust architecture the trust of an asset or user account cannot be established based solely on their physical or network location. Every user, device, and application is treated as potentially hostile until proven otherwise and so they are granted only the minimum level of access necessary to perform their specific tasks, often using just-in-time and just-enough-access methods [7]. A continuous and real time monitoring of network activities guarantees that authorization to various resources is always permitted after authentication. Indeed, this approach uses micro-segmentation for granular policy checks dividing the network into smaller and isolated segments [6].

To the best of our knowledge, this is the first paper that systematically investigates the potential application of the Zero Trust paradigm within the automotive domain. While Zero Trust has been extensively studied and adopted in enterprise networks, cloud computing, and distributed IT infrastructures, its principles have not yet been thoroughly analyzed nor explicitly tailored to the unique architectural, safety, and real-time constraints of modern software-defined vehicles. This work aims to bridge this gap by exploring how Zero Trust concepts—such as continuous authentication, fine-grained authorization, identity-centric security, and runtime verification—can be adapted and integrated into automotive cyber-physical systems.

3. Zero-Trust Software-Defined Vehicles

In this section, we propose the Zero-Trust Software-Defined Vehicles concept and relevant use cases.

3.1. Concept

ZT-SDV enables a secure, auditable, and user-consent-driven marketplace for vehicular applications, transforming the vehicle into a programmable platform capable of installing certified third-party apps even post-sale. Examples include personalized navigation services, advanced driver assistance modules, and infotainment extensions. In the diverse automotive ecosystem, various stakeholders can benefit from enabling this infrastructure: (i) automotive manufacturers designing and producing vehicles; (ii) private users who own one or more vehicles; (iii) public administrations, particularly at the local level; (iv) private companies and developers, providing services in the automotive sector.

As illustrated in Figure 1, ZT-SDV adopts a cloud–fog–edge continuum. Applications can be deployed from the cloud to metropolitan fog nodes and ultimately to on-board ECUs, and dynamically migrated across layers according to workload, latency, and functional requirements. A cloud orchestrator coordinates deployment and migration, while fog orchestrator manage placement at the metropolitan and edge levels.

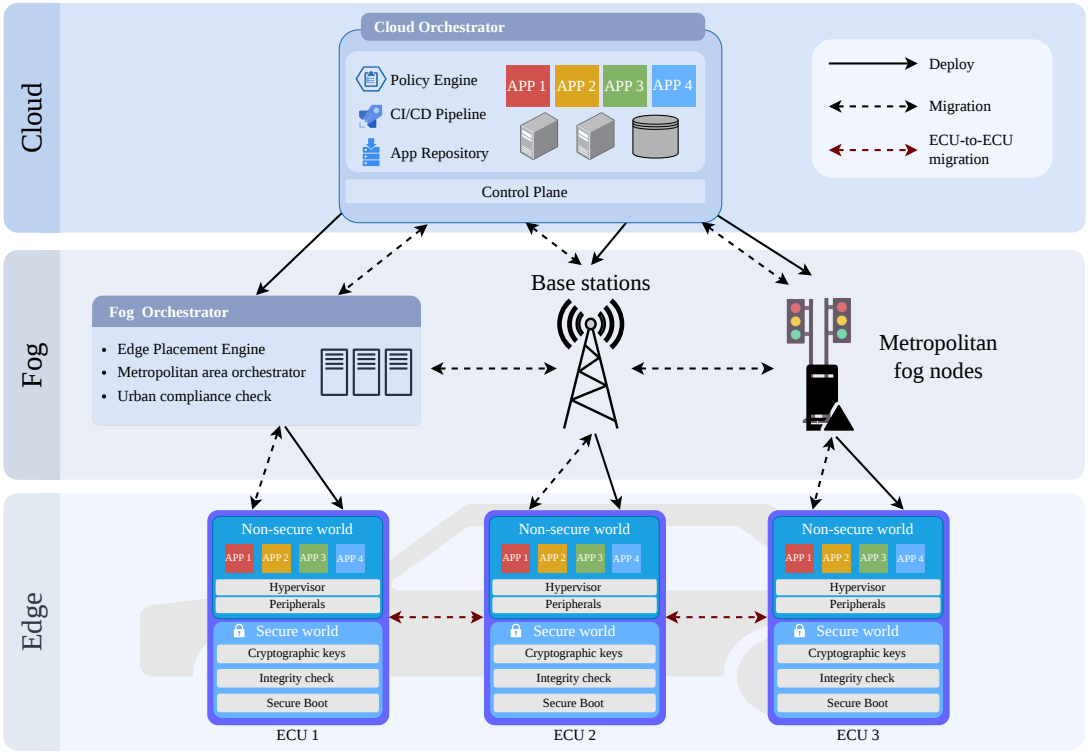

Figure 1: Architecture of Zero-Trust Software-Defined Vehicle and its integration with cloud-edge continuum

At the vehicle level, each ECU runs a hypervisor that enables concurrent execution of multiple applications with different priority levels. The hypervisor enforces strict isolation, provides fine-grained access control, and exposes a uniform API layer that abstracts hardware heterogeneity, allowing developers to write portable applications without architecture-specific adaptations.

To guarantee security and safety, the ECU environment is logically divided into a Non-secure World and a Secure World. Third-party applications execute in the Non-secure World, where they can access only the strictly necessary computational resources and peripherals through controlled interfaces. The Secure World hosts critical security services such as secure boot, cryptographic key management, and integrity verification mechanisms.

Continuous integrity checks, virtualization, and micro-segmentation ensure that even compromised applications cannot affect safety-critical functions. High-priority or safety-related services are granted preferential access to resources, preventing interference with core vehicle functionalities. If a lower-priority task is preempted, it is not explicitly notified; instead, it perceives uninterrupted execution. Avoiding explicit suspension signals prevents unintended information leakage about system state, which is particularly crucial in fault or compromise scenarios.

In this way, ZT-SDV combines flexibility, scalability, and strong isolation guarantees, enabling a secure and dynamically extensible software-defined vehicle architecture.

3.2. Use cases

Thanks to the flexibility, inter-connectivity and modularity of software-defined vehicles, different possible use cases become available. These especially involve the possibility of customization but also modifying vehicle behavior and configuration to re-adapt to different scenarios. In particular, the main aim of ZT-SDVs is not only related to improve users experience, but rather to transform vehicles into comprehensive platforms that contribute to social welfare and utility. For this reason, a very strong relation with the smart city infrastructure, continuously monitoring and communicating with vehicles, will be evident.

Here, a non-exhaustive list of possible use cases is reported. This is because the infrastructure opens up to different and continuing evolving scenarios that include different actor and different entities. Indeed, Figure 2 shows a possible use case diagram summarizing the relations between possible use cases and involved actors.

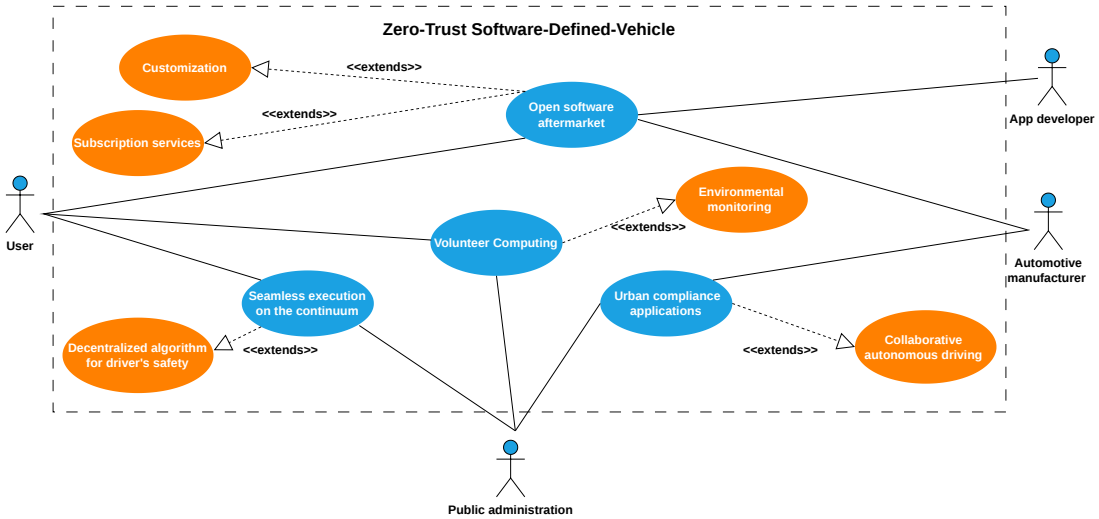

Figure 2: Possible use case diagram of a Zero-Trust Software-Defined vehicle

3.2.1. Open software aftermarket

Through agreements with third-party service providers, the infrastructure could evolve into an open ecosystem similar to mobile app stores. A dedicated marketplace could offer a variety of applications, with distinct characteristics and requirements, which end-users could select based on their vehicle's base capabilities and price preferences. These applications could include features related to safety systems, entertainment systems, or driver-assistance/autonomous driving systems, available through business models based on one-time purchases or subscription services. The installation of certain features may depend on the presence of specific hardware/software resources in the vehicle. A possible approach could involve standardizing the base hardware and software of vehicles, allowing customers to personalize their Cyber Car through application installations, akin to app ecosystems like Android and iOS.

Automotive manufacturers can leverage this infrastructure to optimize production processes, personalize services offered to customers, and make vehicles configurable, upgradable, and extendable in terms of functionality.

3.2.2. Volunteer Computing

In advanced scenarios, vehicle owners could make their vehicles' computational and I/O resources available to public administrations, law enforcement, or private providers for public utility applications in exchange for compensation. For instance, law enforcement agencies could request the temporary installation of a video surveillance application to locate missing persons, executed directly onboard the vehicle to reduce network traffic. Such applications would only be installed on vehicles with adequate computational resources, specific camera characteristics, and, where required, dedicated accelerators.

Alternatively, Cyber Cars could serve as mobile sensors for environmental monitoring experiments, with applications automatically installed and activated in specific geographic areas, subject to owner consent and appropriate incentives. In these cases as well, the applications would only be deployable on vehicles equipped with the required sensors.

It is worth emphasizing that all data are retained within the vehicle, and only the results of processed queries (e.g., whether a person is recognized by onboard cameras) are transmitted externally. This design minimizes data exposure and supports privacy-by-design principles by ensuring that sensitive raw data never leave the system. Furthermore, this collaborative mode is made safe through cryptographic attestation, isolation, and real-time policy enforcement. Apps deployed for such purposes (e.g., crowd-sourced hazard detection or object recognition) are run in restricted virtual machines and may be auto-expiring. Inter-vehicle and vehicle-to-edge communications occur only under mutually authenticated, auditable channels.

3.2.3. Urban compliance applications

Under specific conditions, agreements between automakers and public authorities could mandate the installation of specific applications in certain scenarios. For example, in designated areas, vehicles with adequate resources could be required to run algorithms minimizing internal combustion engine use (or even entirely disabling it in hybrid vehicles), limiting speed, discouraging unsafe driving behaviors, or implementing autonomous driving strategies optimized for safety and efficiency in that area. Similarly, in urban areas with heavy traffic, authorities could enable collaborative autonomous driving, where vehicles are automatically coordinated to optimize traffic flow and manage emergency situations effectively. This approach can also be easily integrated with city infrastructure including smart traffic lights that prioritize particular vehicles and smart parking systems that facilitate finding free spaces and support electric vehicle charging services.

As is already common practice in Limited Traffic Zones (LTZs), non-compliant vehicles could be restricted from accessing such areas. This reflects a widely accepted regulatory approach that not only ensures public safety and reduces environmental impact, but also promotes smoother traffic flow, encourages sustainable urban mobility, and supports the overall well-being of the community.

3.2.4. Seamless execution on the continuum

ZT-SDV enables vehicles to securely participate in distributed computing tasks across fog and edge networks. Integration with the computational continuum also facilitates the execution of complex algorithms distributed between the vehicle and Edge or Cloud nodes. In cases of insufficient onboard computational resources, mandatory algorithms for specific areas or road conditions could still be executed on nearby infrastructure nodes (to minimize latency), ensuring compliance with functional requirements. For example, if a vehicle is entering a special zone area under specific conditions (driver health issues, emergency situation due to pedestrians, other vehicles, or emergency vehicles) an autonomous driving algorithm could be triggered on nearby cloud nodes and take control of the vehicle temporarily in emergencies. This ensures correct compliance with local regulations and, above all, guarantees a high level of safety for users by decoupling sensitive decision logic from the vehicle's own computational resources.

4. Related work

In this section, we review the most relevant literature contributions related to the key pillars of Software-Defined Vehicles (SDVs), and we provide an analysis and comparison against our proposed approach.

Vehicle Integrity. With the increased number of connected devices in the world, securing information and communication exchange represent one of the biggest challenges for developers. This task is more crucial in automotive sector, where compromising one ECU could pose serious problem to the driver's safety. Integrity provides a powerful way of preventing unauthorized users (or not-expert users) from modifying and compromising executing code which can lead to flawed decisions and potential life-endangering actions [8].

A possible solution in this way is ensuring the vehicle's integrity (incorporating the software state of each ECU) from start or during remote software update thanks to a Trusted Software Authority (TSA) who can verify and validate the operations in terms of cryptographic authenticity and correctness [9]. In [10], authors have also highlighted the importance of having a measure of integrity in helping passengers and authorities in identifying an invalid integrity state and eventually disallow the vehicle to move. So, integrity has to be considered a fundamental block in the building process of ZT-SDV which ensure safety and reliability for users and authorities.

Software-Defined Vehicles. In traditional automotive systems, exchange of data between Electronic Control Units (ECUs) is established using signal-based communication. This means the communication structure using protocols such as Controller Area Network (CAN), is static, not flexible and have to be decided beforehand during design phase. Modern vehicles are moving towards a service-oriented architecture (SOA), where software becomes the main character allowing dynamic upgrade capabilities, flexible communication structure using "publish-subscribe" mechanism and easy integration of new devices [11]. Consequently, the structure of modern cars is transforming from Hardware-Defined architecture composed by thousands of Electronic Control Units (ECUs) to a Software-Defined (SD) platform composed by a single or few computational resources. However, in [12] there have been highlighted the possibility of having a hybrid approach in which there could be a static and signal-oriented communication system based on CAN and flexible and service-oriented communication based on Automotive Ethernet.

The flexibility of SOA opens different opportunities for the automotive industry enabling developers and users to easily integrate new functions and personalization and ensuring security and software reliability through remote updates, quality enhancements, and flexible lifecycle management [13]. The included flexibility, together with third-party services however, pose different problems to security measures.

ZTA for connected vehicles The concept of ZTA is relatively new and was initially introduced mainly for network security applications, where the primary goal is to protect confidential and sensitive information. In software-defined vehicles, however, the main risk is not data leakage, but the possibility of executing actions based on compromised information [14]. For this reason, this type of security architecture is fundamental in our system, as it ensures that only trusted and pre-authorized actions (especially for externally provided code) can be executed. While in [15], there have been demonstrated the possibility of implementing a Zero-Trust Architecture to protect the individual components in a connected vehicle using CAN bus, the focus in this paper is to extend the use of this security model also to a SDV using a service-oriented communication system in order to guarantee trustworthiness and integrity during all the platform lifecycle.

In [14], authors proposed a ZT-based intrusion detection system capable of identifying whether a compromised message is circulating within the system. Our approach is aligned with the ZTA paradigm and focuses on acting proactively, limiting the intrusion of compromised external software by continuously enforcing authorization and trustworthiness checks.

In [16], instead, the authors developed a collaborative ZT architecture for vehicular networks in which vehicles exchange minimal trust evidence and apply local, policy-driven decisions to accept or drop communications. Differently from our system, the major approaches of ZTA in vehicles focus on inter-vehicle communications, always emphasizing the primary goal of the architecture's network structure.

A different activity has been developed in [17], where authors implemented and evaluated a Zero-Trust Architecture on an in-vehicle network using a software-in-the-loop setup with resource-constrained hardware, leveraging the gateway ECU to enforce policies, monitor traffic, distribute keys, and segment the network, and using SecOC for message authentication together with Secure Boot to ensure software integrity. Here the focus is mainly related to the network composed by the internal ECUs, but the concept is similar.

In our approach instead, the concept of ZTA involves ECUs, software, programmers, institutions etc. The idea is in fact that every entity that is able to change vehicle behavior must always first be authenticated and authorized, and never trusted by default.

5. Requirements

The described scenarios necessitate a platform specifically engineered to meet critical requirements, crucial for maintaining also the integrity and safety of ZT-SDVs. These requirements extend beyond the vehicle's onboard systems and must also cover software executed on Cloud or Edge nodes within the computational continuum, as these distributed environments increasingly participate in vehicle functionality, data processing, and decision-making.

When dealing with functional requirements, we usually analyze which functions a system must be able to perform determining the system behavior and functionality [18]. Jacobson, Rumbaugh and Booch synthesized the definition as: "A requirement that specifies an action that a system must be able to perform, without considering physical constraints; a requirement that specifies input/output behavior of a system." [19].

By following these definitions, in this section we focus on analyzing and defining all the characteristics that are crucial and essential for completing the system and making it fully operative. Table 1 summarizes the relations between requirements and use cases.

Authorization. In the evolving ecosystem of software-defined vehicles, the possibility to install external applications from marketplaces can lead to unauthorized data access, system misuse, or even control hijacking. For this reason, explicit user authorization is mandatory; no application may run without the vehicle owner's consent. Application execution must be adjustable, such as prioritizing access to accelerators and I/O or limiting computational resource usage. This requirement ensures that the vehicle owners retain sovereignty over the software running on their vehicle, serving as a first line of defense against unauthorized access.

Overrides. ZT-SDVs open the possibility for vehicle owners to participate in community-driven environmental monitoring, traffic analysis, or public safety surveillance efforts. This sharing function however has to be carefully managed. The user, indeed must always be able to monitor resources usage and manually or automatically prevent interference with the vehicle's core safety functions. In this way, resource control together with explicit resource authorization ensure on one hand that only trusted applications can access these resources and at the same time prevent accidental or intentional misuse that could compromise driving safety.

Prioritization. When computational resources are constrained, or under unexpected load, the platform must dynamically re-allocate the load, ensuring that critical real-time functions receive priority access to accelerators and I/O, while limiting or suspending less critical workloads. This capability safeguards the vehicle's operational integrity even under stress or attack scenarios. Fine-grained prioritization is required to maintain a strong balance between functional richness and safety assurance. Critical operations such as braking, steering, and collision avoidance over auxiliary or non-critical applications have to be always prioritized.

Transparency. After explicit authorization, users must be able to verify the actions executed by applications and monitor resource usage. App developers must ensure transparency, offering information about the Cyber Car's base software, installed software, or execution within the computational continuum. This transparency also supports advanced governance scenarios including automatic alerts upon deviations from authorized behavior, and supports lifecycle management practices such as secure software updates and attestation verifications.

Table 1
Mapping between requirements and use cases.

	Open software aftermarket	Volunteer Computing	Urban compliance applications	Seamless execution on the continuum
Authorization	✓	✓		✓
Overrides		✓	✓	✓
Prioritization		✓	✓	✓
Transparency	✓	✓	✓	✓
Attestations	✓	✓		✓
Isolation	✓	✓	✓	✓

Attestations. The highest level of user trust is achieved when the platform can provide demonstrable attestations that the running software originates from verifiable source code. This includes information about software authors, versioning, source code availability, and security audits performed by trusted third parties. For non-expert users who cannot interpret source code or algorithms directly, reliance on communities or certification authorities becomes essential. These entities provide validation and guarantees through certifications or transparent audits, helping users trust the software's security and functionality without needing technical expertise.

Isolation. The coexistence of untrusted or less trusted software alongside safety-critical functions significantly enlarges the attack surface, enabling threats such as buffer-overflow-based code injection or unauthorized access to memory regions and I/O devices. Isolation ensures that each application or subsystem executes within a strictly bounded protection domain, enforced through mechanisms such as virtualization or containerization. At the same time, isolation ensures that an operating system or hypervisor can apply dedicated scheduling and resource management policies to allow applications to share computational resources without violating the temporal and spatial separation requirements of safety-critical workloads.

6. Enabling Technologies

In this section, a list of all enabling technologies, together with a brief description, is presented. Each technology is typically introduced to address one or more requirements; however, a single requirement may be fulfilled by multiple alternative technologies. Table 2 summarizes the relationships between requirements and corresponding identified enabling technologies.

Trusted Execution Environments (TEEs) TEEs are hardware-backed secure enclaves within a processor. They allow critical operations (e.g., key management, policy decision-making, or command validation) to run in isolation from the main OS. Even if the OS is compromised, code in the TEE remains secure and cannot be altered or read externally. TEEs act as the platform root of trust by executing verification routines, enforcing core policies, and mediating secure boot and attestation flows. Typical examples are ARM TrustZone (embedded), Intel SGX (x86), AMD SEV SEV-SNP (confidential VMs).

Lightweight runtime ZT-SDV platforms employ hypervisors or lightweight container engines to enforce strict isolation between software workloads. Each application or third-party service runs in a discrete virtualized environment, unable to access the memory or I/O channels of others. The safe coexistence of multiple applications on a ECU can be achieved through various isolation technologies. For example, the use of containers could allow for the isolation of applications from each other and the host, but it requires, for instance, operating system support. Conversely, technologies like WASM allow for isolation on devices incapable of running a supporting operating system.

Table 2
Mapping between requirements and technologies

	TEE	Lightweight Runtime	Cryptographic Attestation	Reproducible Builds	Micro-Segmentation
Authorization	✓			✓	
Overrides		✓			
Prioritization	✓	✓			✓
Transparency			✓	✓	
Attestations	✓		✓	✓	
Isolation	✓	✓			✓

Cryptographic Attestation Every component, from in-vehicle ECUs to edge-hosted services, must periodically provide cryptographic proof of its identity and of the integrity of the software it is running. Through attestation, the vehicle and its external counterparts can validate that the executing code has not been tampered with and that it matches an approved, known-good baseline (e.g., via signed software measurements). As a result, only entities that are both attested and authorized are allowed to interact with the vehicle, strengthening end-to-end mutual trust across the ecosystem. Looking ahead, this model can evolve into post-quantum cryptographic attestation by adopting quantum-resistant signature schemes so that integrity guarantees remain valid even in a "quantum-capable" threat scenario.

Reproducible Builds Reproducible builds ensure that compiled binaries exactly match their source code, enabling third parties to independently verify what is running on the vehicle. This mechanism helps detect tampering or hidden modifications during the software supply chain. In the ZT-SDV context, reproducibility supports strong provenance guarantees: a user, auditor, or regulator can confirm that an app or control module running on a vehicle was built from a specific, reviewed source code base.

Micro-Segmentation Rather than trusting a unified in-vehicle network, ZT-SDV adopts micro-segmentation dividing the system into tightly controlled trust zones. Communication between zones is explicitly authorized and monitored. For example, a navigation app may access GPS data but not communicate with braking systems. This containment minimizes the potential blast radius of an attack and enforces the principle of least privilege.

7. Possible research trajectories

In this final section, we examine potential research directions that address the implementation of the functionalities and mechanisms introduced above, also considering the challenges that may arise and the corresponding solution approaches. In particular, we devote significant attention to WebAssembly (WASM), an emerging technology that can satisfy several of the requirements needed by our infrastructure and that is increasingly becoming a de facto standard for sandboxing, lightweight virtualization, and flexible software deployment.

Moreover, preliminary results are presented that evaluate the performance of WASM in managing I/O. These values should be interpreted as an initial reference point for the design of the complete infrastructure and, for this reason, they open up opportunities for new and innovative research directions.

7.1. WASM as sandbox mechanisms for MCUs

WebAssembly (WASM) is a portable, low-level bytecode, designed to offer compact representation, serving as an abstraction over modern hardware [20]. High-level languages compile into a universal binary format executed by an embedding runtime, avoiding the need to target specific architectures directly. A WASM binary takes the form of a module, whose definitions may be exported or imported, allowing modules to share resources. To execute, an embedder must instantiate the module, creating a

dynamic representation with its own mutable memory and execution stack. This separation between static binary and dynamic instance enables independent reprogramming at runtime: individual modules can be updated or replaced without a full system reboot or firmware re-flash. Multiple modules can coexist, each executing within a sandboxed environment using fault isolation [21]. Isolation is achieved through WASM's linear memory model, where application storage is restricted to a disjoint byte array, strictly separated from the code space and execution stack. Compiled programs cannot corrupt the execution environment or perform arbitrary jumps, as the bytecode enforces structured control flow.

Although numerous WASM runtimes exist, microcontroller constraints significantly narrow viable options, as suitable runtimes must operate not only within an OS but also in bare-metal configurations. Wasm3 [22] offers an exceptionally small memory footprint through pure interpretation with a minimalist feature set. In contrast, WAMR supports interpretation, Ahead-of-Time (AOT), and Just-In-Time (JIT) compilation, significantly mitigating the performance overhead of pure interpretation.

7.2. Hardware acceleration for WASM

As highlighted by the scientific community [23] and our measurements on Tables 3 and 4, the overhead introduced by WebAssembly runtimes on microcontrollers cannot be neglected, particularly when relying on interpreted execution. While Ahead-of-Time (AOT) compilation can drastically mitigate this performance gap, it compromises the overall portability of the system. By compiling bytecode to machine-specific native code, AOT reduces the flexibility required for dynamic scenarios such as live-migration, where modules must remain architecture-agnostic to move seamlessly between heterogeneous nodes.

To address this trade-off between performance and portability, a recent and emerging approach is the development of hardware accelerators to enhance WASM execution on resource-constrained systems. The foundation of this solution involves general-purpose cores or commercial embedded systems being coupled with specifically designed and prototyped hardware structures (e.g., on FPGAs). These structures are tasked with accelerating the execution of WASM instructions, supporting the main computing system in various ways: by executing only certain phases of the instruction, or, in more drastic cases, by executing the entire WASM instruction on these structures.

In this context, [24] proposes a hardware-software co-design to reduce dispatch overhead in WAMR's fast interpreter. A custom accelerator, tightly coupled with a RISC-V core, pre-fetches the next handler's address in parallel with the CPU's execution of the current opcode. Evaluation on embench-iot shows modest gains (4–11%), highly dependent on the instruction mix: the approach benefits simple integer operations but offers negligible improvement for floating-point or branch-heavy workloads.

Taking a more radical approach, [25] propose a dedicated hardware accelerator designed to execute WASM bytecode directly. Implemented on an FPGA and connected via SPI to the host embedded system, a Raspberry Pi 4 Model B, this architecture utilizes a set of FSMs to natively decode and execute WASM instructions. Experimental results demonstrate a 142-fold speedup compared to traditional software-based embedded runtimes for specific algorithms. However, the accelerator currently supports a limited subset of only 36 instructions.

7.3. Integration of accelerators with CC

The seamless integration of hardware accelerators, such as FPGAs, within the Cloud-Edge Continuum is pivotal for meeting the computational demands of SDVs. However, enabling these resources in a distributed, multi-tenant ecosystem introduces critical challenges such as the ability for multiple distinct applications to concurrently utilize a single shared hardware resource (e.g. an embedded FPGA), to instantiate, dynamically, their own dedicated accelerators.

Consequently, the hardware architecture must be multi-tenant, supporting the coexistence of these diverse acceleration logic within strictly isolated boundaries. To achieve this rigorous isolation without compromising the Zero-Trust model, a Virtual FPGA (vFPGA) architecture can be adopted, as proposed in our related work [26]. In this framework, a vFPGA is defined as a hybrid execution unit, spanning

both the Processing System (PS) and the Programmable Logic (PL). Specifically, the tenant's application logic and the accelerator's orchestration software hosted within a standardized software container on the PS, strictly bound to a dedicated virtual slot on the PL, where the actual hardware accelerator is instantiated.

This architectural split ensures a double layer of security: the software components are logically isolated by the containerization engine, while the accelerators are physically isolated on the FPGA fabric through spatial partitioning. However, in a dynamic SDV ecosystem, hardware accelerators should ideally be "migratable," meaning a specific hardware design could be deployed across heterogeneous FPGA nodes without requiring manual redesign or lengthy re-synthesis processes. While maintaining fixed standard interfaces (e.g., AXI) and preserving source-level compatibility allows for some degree of interoperability, this approach still binds the deployment to specific vendor tools and synthesis times. Maximum flexibility is achieved only when the acceleration logic possesses bitstream-level portability, effectively decoupling the functionality from the underlying physical silicon.

In the literature, this level of abstraction is addressed through the concept of FPGA Overlays [27]. An overlay acts as a virtual intermediate architecture configured on top of the physical fabric, allowing application kernels to be compiled once and deployed anywhere the overlay is present. However, state-of-the-art overlay implementations currently face severe limitations that hinder their adoption in automotive scenarios. Most notably, they suffer from substantial area overhead and low operating clock frequencies [28], often rendering them unusable for high-performance, real-time tasks. Consequently, the development of frequency-optimized, low-overhead overlays represents a crucial research trajectory to fully enable the dynamic migration of hardware-accelerated workloads in the continuum.

Finally, the extension of the Zero-Trust paradigm to the hardware layer requires addressing the integrity of the acceleration logic. In a standard software supply chain, Reproducible Builds are used to guarantee that a binary executable corresponds exactly to a verified source code. However, applying this directly to FPGA bitstreams is notoriously difficult. Without an intermediate abstraction layer, the synthesis process from Hardware Description Language (HDL) to bitstream is inherently non-deterministic. Vendor tools rely on heuristic algorithms (e.g., for Place and Route) and often embed random seeds, meaning that compiling the same hardware design for the same physical board can result in different bitstreams. This variability makes it impossible to distinguish a legitimate build from one tampered with during the supply chain.

FPGA Overlays offer a decisive advantage for verification: by decoupling functional logic from physical implementation, the accelerator becomes a static, architecture-independent binary. This enables simple bitwise verification of the migrated workload, regardless of the target FPGA.

7.4. Virtualized access to I/O peripherals

Consistent with the approach adopted for hardware accelerators, allowing untrusted applications to directly access physical I/O peripherals (such as GPIOs, CAN bus interfaces, or sensors) constitutes a violation of the Zero-Trust model. Direct access would bypass the policy enforcement layer, potentially allowing a malicious module to intercept sensitive signals or actuate safety-critical components without authorization. Therefore, in the ZT-SDV architecture, I/O resources must be strictly virtualized and shared. This paradigm was demonstrated in our previous work [26], where an I/O virtualization layer dynamically maps GPIO pins to applications at runtime. Physical access is strictly mediated by hypervisor-managed hardware mechanisms, enforcing granular per-peripheral policies at the gate level. However, scaling this isolation vehicle-wide poses a key challenge: formally abstracting I/O devices despite their extreme heterogeneity in interfaces, timing, and protocols. A central research direction is finding methods to formally describe these devices, enabling the virtualization layer to expose them uniformly to applications.

In the current WebAssembly ecosystem, this abstraction is handled by the embedder (the runtime), which acts as a proxy exposing secure host functions. While secure, this software mediation introduces latency. To quantify this overhead, we conducted preliminary measurements comparing Native C execution (on Bare-metal, FreeRTOS, and Zephyr), Wasm3 (interpreted), and WAMR (Interpreted and

Table 3
Toggle Benchmark on STM32 F446RE (Cortex-M4 at 180 MHz)

Stack	Runtime	Toggle Frequency	Slowdown vs C
Bare-metal	Native C	36.4 MHz	1.00×
FreeRTOS	Native C	36.4 MHz	1.00×
Zephyr	Native C	36 MHz	1.01×
Bare-metal	Wasm3 (interpreter)	502 kHz	72.5×
FreeRTOS	Wasm3 (interpreter)	564 kHz	64.5×
Zephyr	Wasm3 (interpreter)	523 kHz	69.6×
Zephyr	WAMR (interpreter)	207 kHz	175.8×
Zephyr	WAMR (AOT)	346 kHz	105.2×

Table 4
Toggle Benchmark on STM32 F746ZG (Cortex-M7 at 216 MHz)

Stack	Runtime / Mode	Toggle Frequency	Slowdown vs C
Bare-metal	Native C	110 MHz	1.00×
FreeRTOS	Native C	110 MHz	1.00×
Zephyr	Native C	108 MHz	1.02×
Bare-metal	Wasm3 (interpreter)	908 kHz	121.1×
FreeRTOS	Wasm3 (interpreter)	907 kHz	121.3×
Zephyr	Wasm3 (interpreter)	1.07 MHz	102.8×
Zephyr	WAMR (interpreter)	334 kHz	329.3×
Zephyr	WAMR (AOT)	467 kHz	235.6×

Table 5
Effect of cache configuration (off vs on) on GPIO toggle throughput on the STM32 F746ZG

Stack	Toggle freq. (cache OFF)	Toggle freq. (cache ON)	Speedup cache
Bare-metal (Wasm3)	246 kHz	908 kHz	3.69×
FreeRTOS (Wasm3)	264 kHz	907 kHz	3.43×
Zephyr (Wasm3)	310 kHz	1.07 MHz	3.45×
Zephyr (WAMR interp.)	104 kHz	334 kHz	3.21×
Zephyr (WAMR AOT)	162 kHz	467 kHz	2.88×

Ahead-of-Time) performing a high-frequency GPIO toggle loop. Tests were performed on two platforms: an STM32 Nucleo-F446RE with a Cortex-M4 running at 180 MHz and an STM32 Nucleo-F746ZG with a Cortex-M7 running at 216 MHz. To ensure optimal execution conditions, Instruction Cache (I-Cache) and Data Cache (D-Cache) were explicitly enabled on both architectures.

Table 3 quantifies the cost of virtualization on the STM32 F446RE. Native execution achieves ≈ 36.4 MHz. The transition to WebAssembly introduces a significant penalty: Wasm3 reaches only ≈ 523 kHz, resulting in a ≈ 70x slowdown. Interestingly, WAMR performs worse than Wasm3 in Zephyr, even with AOT compilation (≈ 346 kHz).

On the more powerful platform, the STM32 F746ZG, as illustrated in Table 4, native execution scales efficiently to 110 MHz. Virtualized performance also improves in absolute terms, with Wasm3 reaching ≈ 1.07 MHz and WAMR AOT reaching 467 kHz. However, comparing the two architectures reveals a critical trend regarding the relative impact of virtualization. While the Cortex-M7 offers higher absolute throughput, the slowdown factor worsens paradoxically, widening from 70x on the M4 to over 100x on the M7. This indicates that the runtime infrastructure acts as a bottleneck that fails to scale linearly with the superior hardware capabilities, reinforcing the necessity for the hardware acceleration strategies discussed in 7.2.

Finally, we evaluated the specific impact of the memory hierarchy on the STM32 F746ZG platform. As detailed in Table 5, enabling Instruction and Data Caches (I/D-Cache) along with prefetcher yields a dramatic performance improvement across all virtualized stacks. The speedup factor ranges from $\approx 2.88\times$ (for WAMR AOT) to $\approx 3.69\times$ (for Wasm3 on Bare-metal).

7.5. Integration of reproducible builds and TEE

As highlighted in previous sections, in the context of Software-Defined Vehicles (SDVs), the adoption of reproducible builds becomes crucial to mitigate the risk that potentially malicious third-party software may compromise driver safety. For their rich set of features and the flexibility they provide, WebAssembly (WASM) modules are emerging as a key building block in SDV software architectures. However, the existing literature places limited emphasis on integrating the core principles of reproducible builds with WASM. Most current efforts instead focus on blockchain-based smart contracts, such as Canisters on the Internet Computer, where application logic is compiled to WebAssembly. This asymmetry is not surprising, given the relative novelty of the topic and the fact that reproducible builds have been primarily explored in the broader context of software supply-chain security.

A promising research direction is to extend these ideas to safety-critical SDV scenarios by enabling systematic verification of the source code corresponding to deployed WASM modules. This includes mechanisms that allow independent parties to rebuild modules from source and compare cryptographic hashes of the resulting bytecode with those of in-vehicle binaries. Such capabilities would support stronger guarantees about software provenance and integrity, and reduce the likelihood that compromised toolchains or tampered artifacts introduce unsafe behavior into SDV platforms.

Beyond reproducible builds, Trusted Execution Environments (TEEs) offer hardware support for secure execution in shielded environments, and thus represent a complementary mechanism to enforce security in the design and deployment of distributed applications. In [29], the authors present WATZ, an efficient and secure runtime for trusted execution of Wasm code on Arm TrustZone, together with a lightweight remote attestation system optimized for Wasm applications. Building on this line of work, a core trajectory for SDVs is to adapt these concepts to the automotive domain, combining the sandboxing properties of WASM with TEEs to ensure that code is trustworthy and has not been tampered with. Furthermore, the remote attestation system described in [29] represents a fundamental building block for building an effective and efficient application distribution and verification framework in SDV ecosystems.

7.6. Integration of WASM workloads with CC

Beyond portability and sandboxing, recent work has begun to investigate the live migration of compiled Wasm modules across heterogeneous nodes, running heterogeneous Wasm runtimes in the cloud–edge continuum. Several systems show that it is possible to checkpoint and restore the state of running Wasm modules in a platform-independent format, thus enabling migration between different runtimes and even across architectures with bounded downtime.

For example, in [30] the authors implement a mechanism to checkpoint and restore WebAssembly computations. In particular, checkpoint and restore procedures are encoded as functional bytecode within the Wasm module itself, allowing the execution state to be saved and subsequently resumed after a migration event. This design decouples state management from the underlying runtime, which is essential when targeting heterogeneous execution environments.

The possibility of supporting a flexible lifecycle for Wasm applications has also been investigated in [31]. In that work, the authors propose a Kubernetes-based orchestration architecture for deploying WebAssembly modules on resource-constrained microcontroller units (MCUs) in automotive cyber-physical systems (CPSs). The approach leverages container-style orchestration primitives (scheduling, scaling, relocation) but applies them to Wasm workloads, thereby enabling dynamic placement and reconfiguration of in-vehicle applications. Collectively, these contributions demonstrate that using WebAssembly provides a high degree of flexibility for implementing lightweight workload migration and orchestration,

which is particularly valuable in vehicles hosting multiple, dynamically changing applications. This opens the door to scenarios such as live migration within a vehicle (e.g., between heterogeneous ECUs), between a vehicle and the surrounding infrastructure, or even among different vehicles (for instance, an application tied to a geographic region that migrates from a vehicle leaving the area to another vehicle entering it, while preserving its execution state). In such settings, live migration can be combined with reproducible builds and trusted execution environments to ensure that mobile Wasm workloads remain both verifiable and trustworthy throughout their lifecycle.

8. Conclusions

This paper presented the Zero-Trust Software-Defined Vehicle (ZT-SDV) architecture, a novel paradigm that treats the vehicle as a secure, policy-driven node tightly integrated into the cloud–fog–edge computational continuum. The paper first introduces the problem and related work, then provides an overview of the proposed system, highlighting the key concepts of Zero-Trust Architecture and Software-Defined Vehicles. Subsequently, it details representative use cases, functional requirements, and enabling technologies to clarify the fundamental building blocks of the architecture and to motivate the adoption of a Zero-Trust, software-defined approach in modern automotive ecosystems.

Finally, the paper outlines future research directions with a specific focus on WebAssembly (WASM), discussing how WASM can be integrated into the proposed paradigm and along the Cloud Continuum to shape the future of connected vehicles. In particular, future work will investigate dynamic deployment, migration, and attestation of WASM modules across vehicle, edge, and cloud domains, as well as their interaction with in-vehicle safety-critical components and existing automotive standards. In addition, an important research direction concerns the integration of FPGA-based hardware accelerators within vehicles, enabling the dynamic deployment and migration of accelerators to support computation-intensive and latency-sensitive workloads in a secure and deterministic manner. The role of hardware acceleration for WASM execution will also be explored, studying how WASM runtimes can leverage heterogeneous accelerators to improve performance and energy efficiency while preserving isolation, portability, and Zero-Trust guarantees. Furthermore, future work will investigate I/O virtualization mechanisms that enable secure sharing and remapping of physical sensors, actuators, and communication interfaces across multiple software-defined vehicle functions and WASM modules.

Preliminary benchmarks of different WASM runtimes are also reported as a starting point for future work aimed at exploiting the benefits of WASM in SDVs without compromising performance, including more extensive performance evaluations in realistic vehicular scenarios, security and isolation analyses, and the design of toolchains and development methodologies that make WASM-based SDV applications easier to engineer, verify, and maintain over the entire vehicle lifecycle.

Acknowledgments

The authors would like to express their sincere gratitude to Dr. Salvatore Marchese for his valuable work in performing performance analyses of WASM runtimes on different architectures.
This work was partially supported by the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU through Project “SERICS – Security and Rights in CyberSpace” under Grant PE00000014.

Declaration on Generative AI

During the preparation of this work, the authors used generative AI to paraphrase, reword and improve writing style. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

[1] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka, D. Taibi, Cloud continuum: The definition, IEEE Access 10 (2022) 131876–131886. doi:10.1109/ACCESS.2022.3229185.

[2] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, P. Trunfio, Edge-cloud continuum solutions for urban mobility prediction and planning, IEEE Access 11 (2023) 38864–38874. doi:10.1109/ACCESS.2023.3267471.

[3] G. Merlino, D. Bruneo, F. Longo, A. Puliafito, S. Distefano, Software defined cities: A novel paradigm for smart cities through iot clouds, in: 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015, pp. 909–916. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.174.

[4] W. Zhang, D. Zhang, J. Liu, L. He, A trustworthiness assessment for software based on multi-source evidence fusion, Reliability Engineering & System Safety 266 (2026) 111842. URL: https://www.sciencedirect.com/science/article/pii/S0951832025010427. doi:https://doi.org/10.1016/j.ress.2025.111842.

[5] Y. He, D. Huang, L. Chen, Y. Ni, X. Ma, A survey on zero trust architecture: Challenges and future trends, Wireless Communications and Mobile Computing 2022 (2022) 6476274. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/6476274. doi:https://doi.org/10.1155/2022/6476274. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2022/6476274.

[6] S. Rose, O. Borchert, S. Mitchell, S. Connelly, Zero trust architecture, 2020. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420. doi:https://doi.org/10.6028/NIST.SP.800-207.

[7] S. Syed, Zero trust principles and the evolution of privilege access management architectures, Journal of Computer Science and Technology Studies 7 (2025) 859–865. doi:10.32996/jcsts.2025.7.7.94.

[8] S. Karnouskos, F. Kerschbaum, Privacy and integrity considerations in hyperconnected autonomous vehicles, Proceedings of the IEEE 106 (2018) 160–170. doi:10.1109/JPROC.2017.2725339.

[9] D. Püllen, F. Klement, A. Vinel, S. Katzenbeisser, Ensuring trustworthy automated road vehicles: A software integrity validation approach, in: 2023 IEEE International Automated Vehicle Validation Conference (IAVVC), 2023, pp. 1–8. doi:10.1109/IAVVC57316.2023.10328103.

[10] D. Püllen, N. A. Anagnostopoulos, T. Arul, S. Katzenbeisser, Poster: Hierarchical integrity checking in heterogeneous vehicular networks, in: 2018 IEEE Vehicular Networking Conference (VNC), 2018, pp. 1–2. doi:10.1109/VNC.2018.8628375.

[11] U. Bordoloi, S. Chakraborty, M. Jochim, P. Joshi, A. Raghuraman, S. Ramesh, Autonomy-driven emerging directions in software-defined vehicles, in: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023, pp. 1–6. doi:10.23919/DATE56975.2023.10136910.

[12] M. Rumez, D. Grimm, R. Kriesten, E. Sax, An overview of automotive service-oriented architectures and implications for security countermeasures, IEEE Access 8 (2020) 221852–221870. doi:10.1109/ACCESS.2020.3043070.

[13] M. Traub, A. Maier, K. L. Barbehön, Future automotive architecture and the impact of it trends, IEEE Software 34 (2017) 27–32. doi:10.1109/MS.2017.69.

[14] R. Kaster, D. Ma, Zero-trust enabling intrusion detection protection for software defined vehicles, in: 2024 IEEE 3rd International Conference on Computing and Machine Intelligence (ICMI), 2024, pp. 1–5. doi:10.1109/ICMI60790.2024.10585894.

[15] J. Anderson, Q. Huang, L. Cheng, H. Hu, A zero-trust architecture for connected and autonomous vehicles, IEEE Internet Computing 27 (2023) 7–14. doi:10.1109/MIC.2023.3304893.

[16] B. Dawson, H. Tabatabaee Malazi, A. Kazmi, S. Chaudhry, A zero-trust approach for cooperative vehicular networks: A motorway use case, IEEE Systems Man and Cybernetics Magazine 11 (2025) 11–22. doi:10.1109/MSMC.2024.3441208.

[17] M. E. Shipman, N. Millwater, K. Owens, S. Smith, A zero trust architecture for automotive networks, in: WCX SAE World Congress Experience, SAE International, 2024. URL: https://doi.org/10.4271/

2024-01-2793. doi:https://doi.org/10.4271/2024-01-2793.
[18] M. Glinz, On non-functional requirements, in: 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, pp. 21–26. doi:10.1109/RE.2007.45.
[19] I. Jacobson, G. Booch, J. Rumbaugh, The unified software development process, Addison-Wesley Longman Publishing Co., Inc., USA, 1999.
[20] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, J. Bastien, Bringing the web up to speed with webassembly, SIGPLAN Not. 52 (2017) 185–200. URL: https://doi.org/10.1145/3140587.3062363. doi:10.1145/3140587.3062363.
[21] W. C. Group, Webassembly - security, 2025. URL: https://webassembly.org/docs/security/.
[22] W. C. Group, Wasm3 repository, 2025. URL: https://github.com/wasm3/wasm3.
[23] K. Moron, S. Wallentowitz, Benchmarking webassembly for embedded systems, ACM Trans. Archit. Code Optim. 22 (2025). URL: https://doi.org/10.1145/3736169. doi:10.1145/3736169.
[24] M. Rupp, J. Schröter, S. Wallentowitz, Accelerating webassembly interpreters in embedded systems through hardware-assisted dispatching, in: D. Fey, B. Stabernack, S. Lankes, M. Pacher, T. Pionteck (Eds.), Architecture of Computing Systems, Springer Nature Switzerland, Cham, 2024, pp. 207–220.
[25] J. Kim, R. Kim, J. Oh, S. E. Lee, Hardware-based webassembly accelerator for embedded system, Electronics 13 (2024). URL: https://www.mdpi.com/2079-9292/13/20/3979. doi:10.3390/electronics13203979.
[26] V. A. Bucaria, F. Longo, G. Merlino, F. Restuccia, μ-vf: Enabling virtualization of embedded fpgas, Proc. ACM Meas. Anal. Comput. Syst. 9 (2025). URL: https://doi.org/10.1145/3771581. doi:10.1145/3771581.
[27] A. Brant, G. G. Lemieux, Zuma: An open fpga overlay architecture, in: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, 2012, pp. 93–96. doi:10.1109/FCCM.2012.25.
[28] M. Najem, T. Bollengier, J.-C. Le Lann, L. Lagadec, Extended overlay architectures for heterogeneous fpga cluster management, Journal of Systems Architecture 78 (2017) 1–14. URL: https://www.sciencedirect.com/science/article/pii/S1383762117300413. doi:https://doi.org/10.1016/j.sysarc.2017.06.001.
[29] J. Menetrey, M. Pasin, P. Felber, V. Schiavoni, Watz: A trusted webassembly runtime environment with remote attestation for trustzone, in: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), IEEE, 2022. URL: http://dx.doi.org/10.1109/ICDCS54860.2022.00116. doi:10.1109/icdcs54860.2022.00116.
[30] E. Tinto, L. Marchiori, T. Vardanega, Live migration of compiled wasm modules across the compute continuum, Journal of Systems Architecture 168 (2025) 103532. URL: https://www.sciencedirect.com/science/article/pii/S1383762125002048. doi:https://doi.org/10.1016/j.sysarc.2025.103532.
[31] T. Orlando, M. Arena, L. D'Agati, G. Merlino, F. Longo, Secure and flexible webassembly deployment infrastructure for automotive cyber-physical systems, in: 2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS), 2025, pp. 1–4. doi:10.1109/IOLTS65288.2025.11116815.